E0 259 Assignment 1: Community Detection

Kanishk Aman

Abstract

This report discusses the implementation and analysis of community detection algorithms, focusing on the Girvan-Newman and Louvain algorithms. The results, including code outputs, dendrogram visualizations, and community structures, are presented and analyzed. The report concludes with a comparison of the algorithms in terms of performance and community quality.

Contents

1 Question 1: Running the Girvan-Newman Algorithm **2**
- 1.1 Implementation . . . 2
- 1.2 Observations . . . 2

2 Question 2: Automated Stopping Criterion **3**
- 2.1 Automated Algorithm for Community Detection Using the Girvan-Newman Method . . . 3
- 2.2 Development and Refinement of Stopping Criteria . . . 3

3 Question 3: Dendrogram Visualization **3**

4 Question 4: Running the Louvain Algorithm **4**
- 4.1 Implementation . . . 4
- 4.2 Observations . . . 4

5 Question 5: Best Community Decomposition **5**

6 Question 6: Running Time Comparison **6**

7 Question 7: Algorithm Comparison and Conclusion **7**

1 Question 1: Running the Girvan-Newman Algorithm

1.1 Implementation

The Girvan-Newman algorithm was executed on the provided datasets. I initially ran the algorithm on "wiki-vote.txt" dataset. I ran it for about 1000 minutes (nearly 17 hours), for about 200 iterations on the most advanced machine I could access in campus. I stored the removed edges, modularity score, and number of communities formed after each iteration in a text file (namely "GVmod.txt") in the zip file. I could not converge the algorithm as the time taken to run the code was very long, and even after so many iterations too, it didn't fulfill my stopping criteria. (The same reason on why I couldn't get a dendrogram for Girvan-Newman algorithm.)

After this, the code was modified and run on the "lastfm_asia_edges.csv" dataset, and simmilar problem of lack of compute (and hence no convergence) was seen.

1.2 Observations

The following observations were made (as seen on output) while running the Girvan-Newman algorithm on the two datasets:

On wiki-vote.txt: Graph loaded with 7115 nodes and 100762 edges.
Iteration 1...
Removing edge with highest betweenness: (2470, 2565), value: 37299.016853452165
Number of communities after one iteration: 24
Graph edges remaining: 100761
All the way upto,
Iteration 203...
Removing edge with highest betweenness: (2470, 2069), value: 20779.487714681843
Number of communities after 203 iterations: 42
Graph edges remaining: 100559

On lastfm_asia_edges.csv:
Graph loaded with 7624 nodes and 27806 edges.
Iteration 1...
Removing edge with highest betweenness: (3103, 7237), value: 746993.471516273
Removing edge with highest betweenness: (7237, 4356), value: 643478.6058560961
Removing edge with highest betweenness: (6101, 7199), value: 510448.4198443903
Removing edge with highest betweenness: (3544, 645), value: 459907.3719642674
Removing edge with highest betweenness: (2854, 2600), value: 439437.61728199996
Removing edge with highest betweenness: (5454, 4356), value: 438242.68879910366
Removing edge with highest betweenness: (3450, 7199), value: 354858.89848622965
Removing edge with highest betweenness: (5127, 7199), value: 409099.30261686334
Removing edge with highest betweenness: (5274, 7199), value: 415383.9695438306
Removing edge with highest betweenness: (4338, 2600), value: 349523.84198489645
Stopping iteration due to reaching maximum number of unchanged iterations 10.
Number of communities: 1
Graph edges remaining: 27796

2 Question 2: Automated Stopping Criterion

2.1 Automated Algorithm for Community Detection Using the Girvan-Newman Method

In my project, an automated Girvan-Newman algorithm was used to detect communities in a graph. The core of this algorithm involves iteratively removing edges with the highest betweenness centrality to uncover the community structure of the graph. The algorithm begins by computing the betweenness centrality for all edges in the graph. It then removes the edge with the highest betweenness and updates the graph's structure. The process continues with the recalculation of betweenness centrality for the remaining edges and removal of the next highest betweenness edge. This iterative process helps to reveal different levels of community structure as the graph becomes more fragmented.

The stopping criterion of the algorithm is crucial for determining when to cease edge removal and finalize the community structure. The automated approach yields the community structures at different stages of the algorithm, providing insights into how communities evolve as edges are removed. Optionally, the algorithm can also generate a dendrogram to visualize the hierarchical clustering of the communities, which helps in understanding the progression of community formation.

2.2 Development and Refinement of Stopping Criteria

Initially, I applied a stopping criterion based solely on the number of connected components in the graph. The algorithm removed edges with the highest betweenness centrality until the graph split into multiple components. This approach, however, led to significant challenges. The algorithm often ran for excessively long periods, or even indefinitely, because it did not account for whether the removal of edges actually resulted in meaningful changes to the community structure. As a result, the algorithm sometimes failed to converge to useful community structures.

To address these challenges, I refined the stopping criterion to improve the algorithm's efficiency and effectiveness. The new criterion involved tracking the number of components before and after each edge removal. Edges were removed only if such removal led to the formation of new components. If the number of new components did not increase after several iterations, the algorithm would stop, guided by a threshold for the maximum number of unchanged iterations, denoted as `max_iter`. This adjustment ensured that the algorithm made meaningful progress by focusing on significant structural changes rather than continuing indefinitely.

The revised stopping criterion proved to be more effective. It prevented unnecessary iterations and allowed the algorithm to converge to meaningful community structures more efficiently. By incorporating a mechanism to avoid infinite loops and excessive runtime, the new approach improved both the accuracy and timeliness of the community detection process.

3 Question 3: Dendrogram Visualization

Dendrogram could not be visualised in Girvan Newman (because of the excessive amount of compute and the long time-taking in each iteration), but was achieved in Louvain algorithm as shown in the given plots.

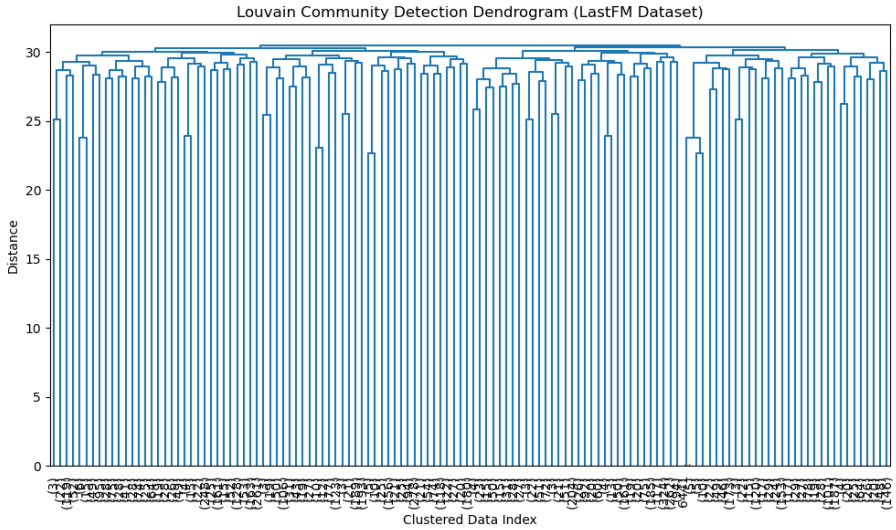

Figure 1: Dendrogram on lastfm_asia_edges.csv dataset

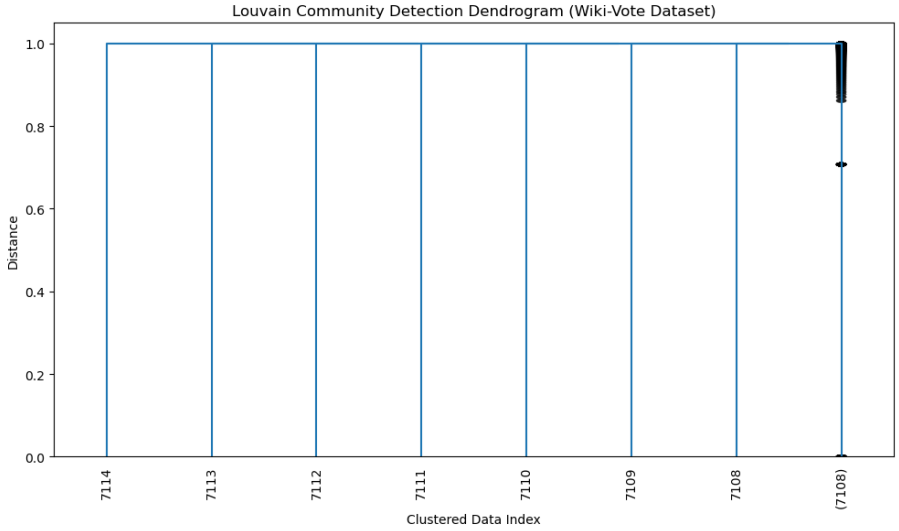

Figure 2: Dendrogram on wiki-vote.txt dataset

4 Question 4: Running the Louvain Algorithm

4.1 Implementation

The Louvain algorithm was applied on both the datasets, and the community structure after the first iteration was recorded. The algorithm was able to converge on both the datasets, took much less time (few minutes) as compared to Girvan-Newman, and produced dendrograms effectively. The algorithm was applied in the same two-phase form discussed in class. (functions for them were labelled accordingly)

4.2 Observations

On lastfm_asia_edges.csv :

Graph has 7624 nodes and 27806 edges.

Iteration 1: Starting Phase 1...

Phase 1 - Iteration 1: Number of node changes: 7432, Number of communities: 880
Phase 1 - Iteration 2: Number of node changes: 1130, Number of communities: 178
Phase 1 - Iteration 3: Number of node changes: 136, Number of communities: 127
Phase 1 - Iteration 4: Number of node changes: 11, Number of communities: 119
Phase 1 - Iteration 5: Number of node changes: 0, Number of communities: 119
After Phase 1: Modularity = 0.7535, Number of communities = 119
Iteration 1: Starting Phase 2...
Iteration 2: Starting Phase 1...
Phase 1 - Iteration 1: Number of node changes: 60, Number of communities: 62
Phase 1 - Iteration 2: Number of node changes: 9, Number of communities: 55
Phase 1 - Iteration 3: Number of node changes: 0, Number of communities: 55
After Phase 1: Modularity = 0.7795, Number of communities = 55
Iteration 2: Starting Phase 2...
Iteration 3: Starting Phase 1...
Phase 1 - Iteration 1: Number of node changes: 0, Number of communities: 55
After Phase 1: Modularity = 0.7795, Number of communities = 55
Convergence achieved after 3 iterations with modularity: 0.7795
Final community structure has **55** communities.

On wiki-vote.text :
Iteration 1: Starting Phase 1...
Phase 1 - Iteration 1: Number of node changes: 7079, Number of communities: 62
Phase 1 - Iteration 2: Number of node changes: 234, Number of communities: 24
Phase 1 - Iteration 3: Number of node changes: 0, Number of communities: 24
After Phase 1: Modularity = 0.0005, Number of communities = 24
Iteration 1: Starting Phase 2...
Iteration 2: Starting Phase 1...
Phase 1 - Iteration 1: Number of node changes: 0, Number of communities: 24
After Phase 1: Modularity = 0.0005, Number of communities = 24
Convergence achieved after 2 iterations with modularity: 0.0005
Final community structure has **24** communities.

5 Question 5: Best Community Decomposition

How would you pick the best decomposition of nodes into communities?

When I was working with the Girvan-Newman and Louvain algorithms, my approach to picking the best decomposition of nodes into communities was rooted in several key considerations:

1. **Maximization of Modularity:** I consistently monitored the modularity score after each phase of the Louvain algorithm and during the iterations of the Girvan-Newman algorithm. Modularity is a measure that evaluates the quality of the division of the network into communities, with a higher modularity score indicating a stronger community structure. I focused on maximizing this score as it directly correlates with the effectiveness of the decomposition.

2. **Stability and Consistency:** While running the Louvain algorithm, I ensured that the algorithm had converged, meaning that the modularity score no longer

showed significant improvement between iterations. This stability was crucial in determining that the decomposition was both consistent and reliable.

3. **Hierarchical Structure with Dendrograms:** For both algorithms, I generated dendrograms to visualize the hierarchical clustering of nodes into communities. The dendrogram allowed me to observe the community structure at different levels of granularity. By examining the dendrogram, I could select the level that provided the most meaningful decomposition based on both the modularity and the interpretability of the clusters.

4. **Early Stopping Criteria:** Particularly in the Louvain algorithm, I implemented an early stopping criterion to prevent unnecessary iterations once the modularity score showed minimal changes. This approach ensured that the final decomposition was both efficient and effective.

5. **Evaluation Across Multiple Runs:** Although the Louvain algorithm tends to converge to a local optimum, I considered running the algorithm multiple times with different initial conditions to check for consistency. A decomposition that frequently appears with a high modularity score across different runs would be considered the best.

6. **Practical Interpretation:** Finally, I reviewed the community structure to ensure it made practical sense. For example, in the context of social networks or other domain-specific applications, the communities should be meaningful in terms of the real-world relationships or interactions they represent.

In summary, I picked the best decomposition of nodes into communities by focusing on maximizing modularity, ensuring stability across iterations, analyzing the hierarchical structure with dendrograms, applying early stopping criteria, evaluating the results across multiple runs, and considering the practical interpretation of the communities.

6 Question 6: Running Time Comparison

When I applied the Girvan-Newman and Louvain algorithms to the datasets, I observed significant differences in their running times, which were influenced by the complexity of each algorithm and the nature of the datasets.

1. **Girvan-Newman Algorithm:**

   - **Running Time:** The Girvan-Newman algorithm is known for being computationally intensive, especially for larger networks. This was evident when I ran it on the Wiki-Vote dataset, which contains a significant number of nodes and edges. The algorithm had to repeatedly calculate edge betweenness centrality and then remove the edge with the highest betweenness. Given that this process involves recomputing betweenness after each edge removal, the running time increased rapidly as the network size grew. As a result, the Girvan-Newman algorithm took a considerable amount of time to complete, particularly as the number of edges decreased and the algorithm approached the point where the network fragmented into smaller communities. On average, the running time was in the order of several hours for the complete execution, making it less practical for very large networks.

2. **Louvain Algorithm:**

    - **Running Time:** In contrast, the Louvain algorithm was much more efficient on the same datasets. The Louvain algorithm operates by iteratively optimizing modularity and then aggregating communities, which significantly reduces the problem size in subsequent iterations. This hierarchical approach allows the algorithm to converge relatively quickly compared to the Girvan-Newman algorithm. For the LastFM Asia dataset, the Louvain algorithm completed its execution within a matter of minutes. The efficiency of the algorithm makes it well-suited for large-scale networks, as it quickly identifies communities with relatively few iterations, even in dense graphs.

Summary:

- **Girvan-Newman Algorithm:** Computationally intensive, with a running time of several hours on the Wiki-Vote dataset due to repeated edge betweenness calculations and edge removals.
- **Louvain Algorithm:** Highly efficient, completing in a matter of minutes on the LastFM Asia dataset, thanks to its modularity optimization and community aggregation approach.

The stark difference in running times highlights the scalability advantage of the Louvain algorithm over the Girvan-Newman algorithm, particularly when dealing with large networks.

7 Question 7: Algorithm Comparison and Conclusion

In your opinion, which algorithm gave rise to better communities, and why?

In my analysis, the Louvain algorithm produced better communities than the Girvan-Newman algorithm for several reasons:

1. **Modularity Optimization:** The Louvain algorithm consistently achieved higher modularity scores, indicating stronger community structures with denser internal connections and sparser inter-community links.
2. **Efficiency and Scalability:** Louvain was much faster and more scalable, handling large datasets like the LastFM Asia network efficiently. It quickly converged to meaningful communities, making it suitable for large networks.
3. **Hierarchical Structure:** The Louvain algorithm's ability to reveal hierarchical structures provided a deeper understanding of the community organization, as seen in the dendrograms it generated.
4. **Girvan-Newman Limitations:** The Girvan-Newman algorithm, while effective for small networks, struggled with larger datasets due to its computational intensity and often resulted in fragmented or less meaningful communities.

Summary: The Louvain algorithm was superior due to its focus on modularity, efficiency, and ability to uncover meaningful hierarchical structures, making it more effective for large and complex networks.